



Java Programming

9-2

JDBC Basics



Objectives

This lesson covers the following topics:

- JDBC Data Types
- Programming with JDBC PreparedStatement
- Programming with JDBC CallableStatement
- Reading MetaData from Database



JDBC Data Types:

- The Java Language has a data type system, for example Boolean, int, long, float, double, string.
- Oracle Database systems also have a data type system, such as int, char, varchar2, text, blob, clob.
- The Oracle JDBC Drivers support standard JDBC data type and Oracle-specific datatypes also.
- The JDBC driver can convert a Java data type to the appropriate database type and vice versa.

JDBC Data Types:

- The JDBC type system controls the conversion between Oracle data types and Java language types and objects.
- The JDBC types are modeled on the SQL-92 and SQL-99 types.

JDBC Data Types:

JDBC Data Types list sample:

SQL Datatypes	JDBC Typecodes	Standard Java Types	Oracle Extension Java Types
	STANDARD JDBC 1.0 TYPES:		
CHAR	java.sql.Types.CHAR	java.lang.String	oracle.sql.CHAR
VARCHAR2	java.sql.Types.VARCHAR	java.lang.String	oracle.sql.CHAR
LONG	java.sql.Types.LONGVARCHAR	java.lang.String	oracle.sql.CHAR
NUMBER	java.sql.Types.NUMERIC	java.math.BigDecimal	oracle.sql.NUMBER
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE

A full list of JDBC Data Types can be found at https://docs.oracle.com/cd/B19306_01/java.102/b14188/datamap.htm

The PreparedStatement Object

- For a Database, calling a precompiled SQL statement is more efficient than repeatedly calling the same SQL statement.
- The setter methods (setShort, setString, and so on) for setting IN parameter values must specify types that are compatible with the defined SQL type of the input parameter.
- It Extends the Statement interface.

The PreparedStatement Object

- A prepared statement is a precompiled SQL statement .
- The SQL statement is parsed only once.
- The PreparedStatement interface extends the Statement interface to add the capability of passing parameters inside of a statement.
- If the same SQL statements are executed multiple times, use a PreparedStatement object.

The PreparedStatement Object

```
PreparedStatement pstmt = conn.prepareStatement(sqlString);
```

- Example 1:

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE EMPLOYEES  
SET SALARY = ? WHERE ID = ?");  
pstmt.setBigDecimal(1, 100000.00)  
pstmt.setInt(2, 110592)
```

The PreparedStatement Object

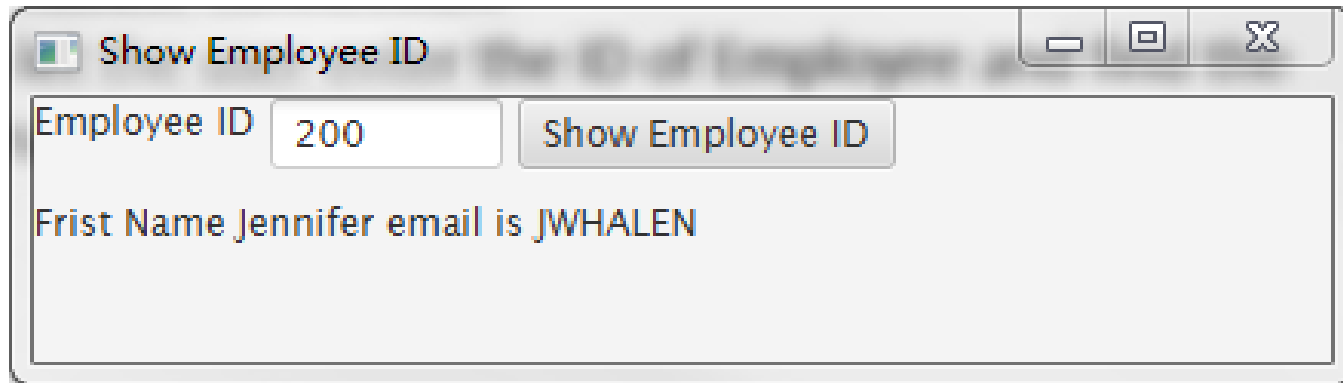
```
PreparedStatement pstmt = conn.prepareStatement(sqlString);
```

- Example 2

```
PreparedStatement prepStmt = con.prepareStatement("SELECT Department_id  
FROM DEPARTMENT WHERE department_name = ?");  
prepStmt.setString(1, "Marketing");  
ResultSet rs = prepStmt.executeQuery();
```

PreparedStatement: Retrieving Data

- The following is an example that demonstrates a JavaFx application establishing a database connection.
- The Program prompts the user to enter the ID of Employee to find the Employee Name and email address.



PreparedStatement: Retrieving Data

```
import java.sql.*;
import java.math.BigDecimal;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import oracle.jdbc.pool.OracleDataSource;
```

PreparedStatement: Retrieving Data

```
public class SearchEmployee extends Application {  
  
    private Connection conn;  
    private PreparedStatement pstmt;  
    private ResultSet rset;  
    private TextField tfid = new TextField();  
    private Label lblResult = new Label();  
}
```

PreparedStatement: Retrieving Data

@Override

```
public void start(Stage primaryStage) {  
    initializeDatabase();  
    Button btShowID = new Button("Show Employee Email");  
    HBox hBox = new HBox(5);  
    hBox.getChildren().addAll(new Label("Employee ID"), tfid, btShowID);  
    VBox vBox = new VBox(10);  
    vBox.getChildren().addAll(hBox, lblResult);  
    tfid.setPrefColumnCount(6);  
    btShowID.setOnAction(e -> showResult());  
    Scene scene = new Scene(vBox, 400, 100);  
    primaryStage.setTitle("Show Employee ID");  
    primaryStage.setScene(scene);  
    primaryStage.show();  
}
```

PreparedStatement: Retrieving Data

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}
public void initializeDatabase() {
    try {
        OracleDataSource ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:dfot/dfot@localhost:1521:xe");
        conn = ods.getConnection();
        String queryID = "select email,First_name from Employees "
            + "where employee_id = ?";
        pstmt = conn.prepareStatement(queryID);
    } catch (Exception e) {e.printStackTrace();}
}
```

PreparedStatement: Retrieving Data

```
private void showResult() {  
    String id = tfid.getText();  
    try {  
        pstmt.setBigDecimal(1,new BigDecimal(id));  
        rset = pstmt.executeQuery();  
        if (rset.next()) {  
            String email = rset.getString(1);  
            String firstName = rset.getString(2);  
            lblResult.setText("Frist Name " + firstName + " email is " + email);  
        } else { lblResult.setText("Try again !No Employee information for the ID "+id);  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```


PreparedStatement: DML Statement

- The PreparedStatement interface is an extended Statement interface.
- We would use the Statement interface to execute static SQL statements that do not contain any parameters.
- We can use the PreparedStatement interface to execute a precompiled SQL statement with or without parameters.

PreparedStatement: DML Statement

- A PreparedStatement object is created using the `prepareStatement` method in the `Connection` interface.
- For example, the following code creates a PreparedStatement for a SQL delete statement:

```
DELETE FROM Employees where employee_id=?
```

- Select statement:

```
Select job_title from jobs where job_id=? Order by ?
```

- Insert statement:

```
Insert into regions(region_id,region_name) values (?,?)
```

PreparedStatement: SQL Insert

- The Insert statement has two question marks which work as the placeholders for parameters region_id, region_name in the regions table.
- The setter methods setX(setShort, setString and so on) for setting IN parameter values.

Insert into regions(region_id,region_name) values (?,?)

setX(int columnIndex, X value);

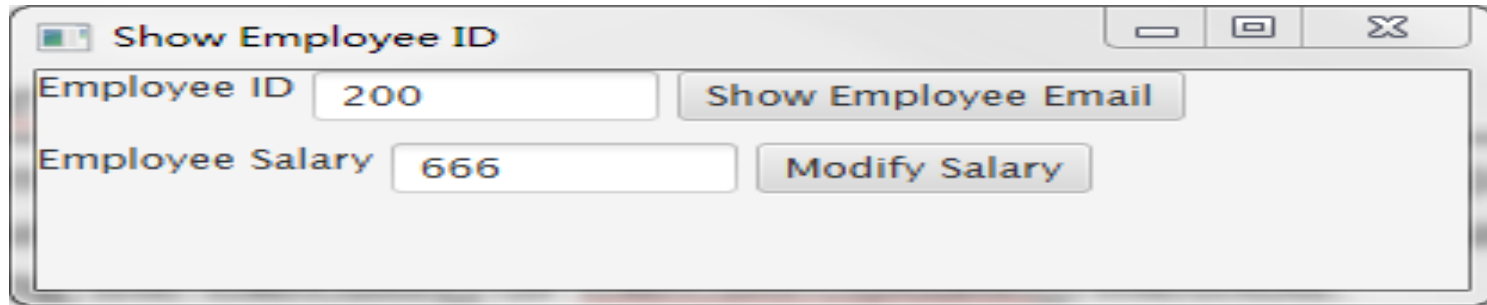
PreparedStatement : SQL Insert

- Where X is the type of the parameter and the columnIndex is the index of the parameter in the statement.
- Execute the PreparedStatement:
`pstmt.executeUpdate();`

PreparedStatement: Update Data

- This example uses the prepared statement to pass parameters.
- After setting the parameters, you can execute the prepared statement by invoking the `execute()` or the `executeUpdate()` method.
- In the example, the program passes the salary and `employee_id` parameters to the update sql statement.
- The sql statements will be compiled before they are sent to the database.

PreparedStatement: Update Data



```
import java.sql.*;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import oracle.jdbc.pool.OracleDataSource;
```

PreparedStatement: Update Data

```
public class EmployeeSalary extends Application {  
    private Connection conn;  
    private PreparedStatement pstmt;  
    private Statement stmt;  
    private ResultSet rset;  
    private TextField tfid = new TextField();  
    private TextField tfSalary = new TextField();  
    private Label lblResult = new Label();  
}
```

PreparedStatement: Update Data

@Override

```
public void start(Stage primaryStage) {  
    initializeDatabase();  
    Button btShowID = new Button("Show Employee Email");  
    Button btShowSalary = new Button("Modify Salary");  
  
    HBox hBox1 = new HBox(5);  
    hBox1.getChildren().addAll(new Label("Employee ID"), tfid, btShowID);
```


PreparedStatement: Update Data

```
HBox hBox2 = new HBox(5);
    hBox2.getChildren().addAll(new Label("Employee Salary"), tfSalary, btShowSalary);
    VBox vbox = new VBox(10);
    vbox.getChildren().addAll(hBox1, hBox2, lblResult);
    tfid.setPrefColumnCount(8);
    tfSalary.setPrefColumnCount(8);
    btShowID.setOnAction(e -> showResult());
    btShowSalary.setOnAction(e -> changeSalary());
    Scene scene = new Scene(vbox, 400, 100);
    primaryStage.setTitle("Show Employee ID");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

PreparedStatement: Update Data

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}
public void initializeDatabase() {
    try {
        OracleDataSource ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:dfot/dfot@localhost:1521:xe");
        conn = ods.getConnection();
        stmt = conn.createStatement();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

PreparedStatement: Update Data

```
private void showResult() {  
    String id = tfid.getText();  
    System.out.println("employee id="+id);  
    try {  
        stmt = conn.createStatement();  
        String queryID = "select email,First_name,salary from Employees "  
            + "where employee_id = '" + id + "'";  
        rset = stmt.executeQuery(queryID);  
        if (rset.next()) {  
            String email = rset.getString(1);  
            String firstName = rset.getString(2);  
            tfSalary.setText(rset.getBigDecimal(3).toString());  
            lblResult.setText("Frist Name " + firstName + " email is " + email);  
        }  
    }  
}
```

PreparedStatement: Update Data

```
else {  
    lblResult.setText("No Result!");  
}  
rset.close(); stmt.close();  
//conn.close();  
} catch (SQLException e) {  
    e.printStackTrace();  
}  
}
```

PreparedStatement: Update Data

```
private void changeSalary() {  
    try{  
        int id = Integer.parseInt(tfid.getText());  
        int salary=Integer.parseInt(tfSalary.getText());  
        pstmt = conn.prepareStatement("UPDATE employees SET salary = ?  
where employee_id =?");  
        stmt = conn.createStatement();  
        String queryID = "select email,First_name,salary from Employees "  
            + "where employee_id = '" + id + "'";  
        rset = stmt.executeQuery(queryID);
```

PreparedStatement: Update Data

```
pstmt.setInt(1,salary);  
    pstmt.setInt(2,id);  
    pstmt.execute();  
    rset.close();stmt.close();
```

```
    }catch (NumberFormatException e){ System.out.println("employee  
id=");  
        } catch (SQLException e) {  
  
            e.printStackTrace();  
        }  
    }  
}
```

CallableStatement

- The CallableStatement interface is designed to execute SQL stored procedures.
- The Oracle JDBC Drivers support Java and PL/SQL stored procedures.
- The procedures may have IN, OUT or IN/OUT parameters.
- IN parameter values are set using the set methods inherited from PreparedStatement.
- OUT parameters must be registered before executing the stored procedure.
- The result can be retrieved via get methods.

Oracle Stored Procedure Sample

- As with PreparedStatement objects, input parameters must be set with the setType method, where TYPE is the appropriate data type for the parameter.
- Output parameters are defined with the registerOutParameter method of a CallableStatement object.

Oracle Stored Procedure Sample

Assume an Oracle stored procedure exists for a given employee id, and returns the salary and first name.

```
CREATE OR REPLACE PROCEDURE getEmployeeSalary(emp_id IN
NUMBER,sal OUT Number,f_name OUT varchar2)
AS
BEGIN
SELECT salary,first_name INTO sal,f_name
FROM EMPLOYEES
WHERE EMPLOYEE_ID=emp_id;
end;
```

CallableStatement Object

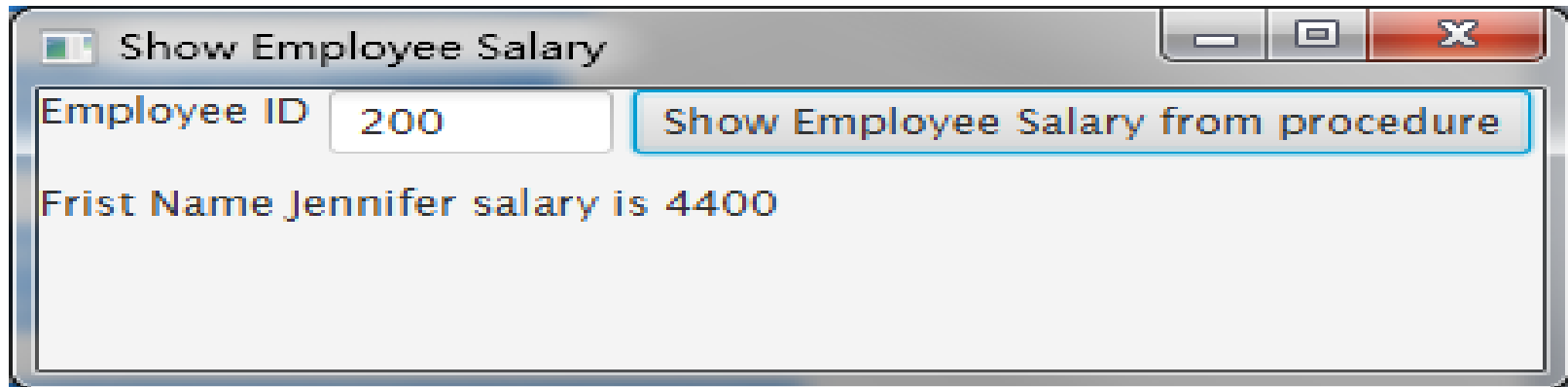
- A JDBC CallableStatement object is created by invoking the callableStatement method with the name of the procedure and question marks (“?”) to represent the input and output parameters.
- The CallableStatement interface is designed to execute SQL-stored procedures.

CallableStatement Object

- The procedures may have IN, OUT or IN/OUT parameters.
- An IN parameter receives a value passed to the procedure when it is called.
- An OUT parameter returns a value after the procedure is completed, but it doesn't contain any value when the procedure is called.
- An IN/OUT parameter contains a value passed to the procedure when it is called and returns a value after it is completed.

CallableStatement Object

- For example, the following procedure in Oracle PL/SQL has IN parameter emp_id, OUT parameter sal, OUT parameter f_name.



- getEmployeeSalary(emp_id IN NUMBER,sal OUT Number,f_name OUT varchar2)

CallableStatement Example

```
import java.sql.*;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
import oracle.jdbc.pool.OracleDataSource;

public class CallableEmployee extends Application {
    private Connection conn;
    private CallableStatement cstmt;
    private TextField tfid = new TextField();
    private Label lblResult = new Label();
```

CallableStatement Example

```
@Override
public void start(Stage primaryStage) {
    initializeDatabase();
    Button btShowID = new Button("Show Employee Salary from
procedure");
    HBox hBox = new HBox(5);
    hBox.getChildren().addAll(new Label("Employee ID"), tfid, btShowID);
    VBox vBox = new VBox(10);
    vBox.getChildren().addAll(hBox, lblResult);
    tfid.setPrefColumnCount(6);
    btShowID.setOnAction(e -> showResult());
    Scene scene = new Scene(vBox, 400, 100);
    primaryStage.setTitle("Show Employee Salary");
    primaryStage.setScene(scene);
    primaryStage.show();
}
```

CallableStatement Example

```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    launch(args);
}

public void initializeDatabase() {
    try {
        OracleDataSource ods = new OracleDataSource();
        ods.setURL("jdbc:oracle:thin:dfot/dfot@localhost:1521:xe");
        conn = ods.getConnection();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

CallableStatement Example

```
private void showResult() {  
    lblResult.setText("");  
    String sql = "{call getEmployeeSalary(?,?,?)}";  
    try {  
        int id = Integer.parseInt(tfid.getText());  
        cstmt = conn.prepareCall(sql);  
        cstmt.setInt(1, id);  
        cstmt.registerOutParameter(2, Types.DOUBLE);  
        cstmt.registerOutParameter(3, Types.VARCHAR);  
        cstmt.execute();  
        String salary = cstmt.getBigDecimal(2).toString();  
        String firstName = cstmt.getString(3);  
        lblResult.setText("Frist Name " + firstName + " salary is " + salary);  
    }  
}
```


CallableStatement Example

```
catch (SQLException e) {  
    lblResult.setText("wrong Employee id, please try again!");  
} catch (NumberFormatException ne){  
    lblResult.setText("wrong Employee id, please try again!");  
}  
}  
}
```

Reading Metadata

- In some cases, the application may require to read the metadata.
- For example, the DatabaseMetaData and ResultSetMetaData.
- JDBC provides the DatabaseMetaData interface for retrieving Comprehensive information about the database as a whole.

Reading Metadata

- DatabaseMetaData is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS) in combination with the driver based on JDBC™ technology ("JDBC driver") that is used with it.
- The database metadata such as username, table name and database URL can be accessed by the DatabaseMetaData interface.
- You can use the ResultSetMetaData object to get information about the types and properties of the columns.

Database MetaData

- The Connection object provides access to database metadata information that describes the capabilities of the Oracle Database.
- To get an instance of the DatabaseMetaData for a database, use the getMetaData method defined on a Connection Object.

```
DatabaseMetaData dbmd = conn.getMetaData();
```

Database MetaData

- You can invoke the methods defined in DatabaseMetaData .
- `getDriverName()`
 - Retrieves the name of this JDBC driver.
- `getSchemas()`
 - Retrieves the schema names available in this database.
- `getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)`
 - Retrieves a description of the tables available in the given catalog.

Result MetaData

- ResultMetaData object can be used to get information about the types and properties of the columns in a ResultSet object.
- The code fragment on the following slide :
 - creates the ResultSet object rs
 - creates the ResultSetMetaData object rsmd
 - uses rsmd to find out how many columns rs has, and whether the first column in rs can be used in a WHERE clause.

Result MetaData

```
ResultSet rs = stmt.executeQuery("SELECT JOB_ID, JOB_TITLE FROM  
JOBS");  
    ResultSetMetaData rsmd = rs.getMetaData();  
    int numberOfColumns = rsmd.getColumnCount();  
    boolean b = rsmd.isSearchable(1);
```

Summary

In this lesson, you should have learned:

- JDBC Data Types
- Programming with JDBC PreparedStatement
- Programming with JDBC CallableStatement
- Reading MetaData from Database



